

TP3 – Performance d'un classifieur

Réalisé par : BENSLIMANE Zahra

Chargé de TP : Flavien Lebrun

Chargée de cours : Catrine Achard

Résumé : L'objectif du TP est de détecter des pixels de teinte chair afin d'accélérer un algorithme de détection de visages. Pour cela on va travailler avec les images de la base Pratheepan Dataset (W.R. Tan, C.S. Chan, Y. Pratheepan and J. Condell (2012) "A Fusion Approach for Efficient Human Skin Detection", IEEE Transactions on Industrial Informatics)

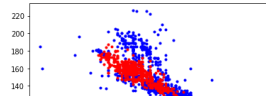
I. Chargement et visualisation des données

```
# Chargement des librairies
import numpy as np
import matplotlib.pyplot as plt
import math
```

On a récupéré dans X_train, y_train, les pixels des 26 premières images et dans X_test, y_test, ceux des 4 dernières images. Seul un pixel sur 2000 a été conservé pour avoir des temps de calcul raisonnables. X_ est composé des composantes chromatiques Cb, Cr de chaque pixel et y_ est composé de la classe du pixel, 1 si couleur peau, 0 sinon

```
#Observer les images dans l'explorateur de fichier puis charger les données et visualiser
[X_train, y_train, X_test, y_test] = np.load('TP3.npy', allow_pickle=True)
#Pixel peau
T_train = X_train[np.where(y_train==1),:]
T_train = np.reshape(T_train, (T_train.shape[1], T_train.shape[2] ))
#Pixel non peau
F_train = X_train[np.where(y_train==0),:]
F_train = np.reshape(F_train, (F_train.shape[1], F_train.shape[2] ))
plt.plot(F_train[:,0], F_train[:,1], 'b')
plt.show
plt.plot(T_train[:,0], T_train[:,1], 'r')
plt.show
```

<function matplotlib.pyplot.show>



Questions Combien y a-t-il de pixels de teinte chair ? de teinte non chair ? Quelle est la dimension des données ?

```
print("X_train.shape = ", X_train.shape )
print("y_train.shape = ", y_train.shape)
print("X_test.shape = ", X_test.shape)
print("y_test.shape = ", y_test.shape)

print("\nT_train.shape = ", T_train.shape, "#Pixel peau : Deux composantes Cb et Cr")
print("F_train.shape = ", F_train.shape, " #Pixel non peau : : Deux composantes Cb et Cr")

print("\nNombre de pixels de la classe chair : ", T_train.shape[0])
print("Nombre de pixels de la classe non chair : ", F_train.shape[0])

print("La dimension des données = ", 2) # Cb et Cr
```

```
X_train.shape = (2378, 2)
y_train.shape = (2378,)
X_test.shape = (284, 2)
y_test.shape = (284,)

T_train.shape = (639, 2) #Pixel peau : Deux composantes Cb et Cr
F_train.shape = (1731, 2) #Pixel non peau : : Deux composantes Cb et Cr

Nombre de pixels de la classe chair : 639
Nombre de pixels de la classe non chair : 1731
La dimension des données = 2
```

II. Modélisation de la densité de probabilité a priori de la teinte chair par une loi normale 2D avec des dimensions décorrélées

Comme il est difficile de modéliser tout ce qui n'est pas teinte chair, on décide de travailler avec une seule classe, la teinte chair, dont on estime la densité de probabilité a priori. $p(x|chair)$.

a. Estimation de la densité de probabilité a priori des pixels de teinte chair

On modélise la densité de probabilité a priori des pixels de teinte chair $p(x|chair)$, $x = [Cb Cr]$ sur la base d'apprentissage par une loi normale 2D avec des dimensions décorrélées :

https://colab.research.google.com/drive/14QMxGy_4k8WlaGwJCZVgSQRCPKdVvIpqscroITo?E5WMHWIBXzoQ

1/11

10/04/2022 20:27

TP3_ML_Benslimane.iyyrb - Collaboratory

```
print("Pour faire le produit des probabilités, on considère que les classes sont indépendantes")

len(p1_train) = 639

Pour un pixel x de teinte chair donnée, la dimension de  $p(x|chair)$  est de : 1
La dimension du vecteur p1_train = 639
Pour faire le produit des probabilités, on considère que les classes sont indépendantes
```

b. Classification

Afin de classer les pixels de test X_test comme teinte chair ou non, on estime la valeur de $p(x|chair)$ en ces points que l'on stocke dans P1_test. Pour réaliser la classification, on utilise P1_test en utilisant comme seuil initial la valeur moyenne de P1_train.

Sur la base de test, estimer :

- TP : le nombre de vrai positif
- TN : le nombre de vrai négatif
- FP : le nombre de faux positif
- FN : le nombre de faux négatif

puis la sensibilité et la spécificité. Quel est le taux de bonne reconnaissance ?

```
print("X_test.shape = ", X_test.shape)
print("y_test.shape = ", y_test.shape)
# Estimation de la la valeur de  $p(x|chair)$  des points test
p1_test = []
for i in range(X_test.shape[0]):
    p1_test.append(norml(X_test[i, 0], mCb, σCb) * norml(X_test[i, 1], mCr, σCr))
print("p1_test.shape = ", len(p1_test))
```

```
X_test.shape = (284, 2)
y_test.shape = (284,)
p1_test.shape = 284
```

```
def classifyData(estimatedProba, threshold):
    binaryClassif = []
    for i in range(len(estimatedProba)):
        binaryClassif.append(1 if estimatedProba[i] >= threshold else 0)
    return binaryClassif
```

```
# Classification : On seuille P1_test : Binarisation
seuil_initial = np.mean(p1_train) # on seuille P1_test en utilisant comme seuil initial 1
print("Le seuil initial = la valeur moyenne de P1_train = ", seuil_initial)
binaryClassif = classifyData(p1_test, seuil_initial)
#print(p1_test) #print(binaryClassif)
```

```
Le seuil initial = la valeur moyenne de P1_train = 0.00881445019677562
```

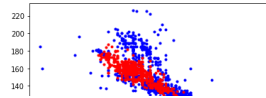
https://colab.research.google.com/drive/14QMxGy_4k8WlaGwJCZVgSQRCPKdVvIpqscroITo?E5WMHWIBXzoQ

4/11

10/04/2022 20:27

TP3_ML_Benslimane.iyyrb - Collaboratory

<function matplotlib.pyplot.show>



Questions Combien y a-t-il de pixels de teinte chair ? de teinte non chair ? Quelle est la dimension des données ?

```
print("X_train.shape = ", X_train.shape )
print("y_train.shape = ", y_train.shape)
print("X_test.shape = ", X_test.shape)
print("y_test.shape = ", y_test.shape)

print("\nT_train.shape = ", T_train.shape, "#Pixel peau : Deux composantes Cb et Cr")
print("F_train.shape = ", F_train.shape, " #Pixel non peau : : Deux composantes Cb et Cr")

print("\nNombre de pixels de la classe chair : ", T_train.shape[0])
print("Nombre de pixels de la classe non chair : ", F_train.shape[0])

print("La dimension des données = ", 2) # Cb et Cr
```

```
X_train.shape = (2378, 2)
y_train.shape = (2378,)
X_test.shape = (284, 2)
y_test.shape = (284,)

T_train.shape = (639, 2) #Pixel peau : Deux composantes Cb et Cr
F_train.shape = (1731, 2) #Pixel non peau : : Deux composantes Cb et Cr

Nombre de pixels de la classe chair : 639
Nombre de pixels de la classe non chair : 1731
La dimension des données = 2
```

II. Modélisation de la densité de probabilité a priori de la teinte chair par une loi normale 2D avec des dimensions décorrélées

Comme il est difficile de modéliser tout ce qui n'est pas teinte chair, on décide de travailler avec une seule classe, la teinte chair, dont on estime la densité de probabilité a priori. $p(x|chair)$.

a. Estimation de la densité de probabilité a priori des pixels de teinte chair

On modélise la densité de probabilité a priori des pixels de teinte chair $p(x|chair)$, $x = [Cb Cr]$ sur la base d'apprentissage par une loi normale 2D avec des dimensions décorrélées :

https://colab.research.google.com/drive/14QMxGy_4k8WlaGwJCZVgSQRCPKdVvIpqscroITo?E5WMHWIBXzoQ

2/11

10/04/2022 20:27

TP3_ML_Benslimane.iyyrb - Collaboratory

```
def cf_matrix(y_test, y_predicted):
    # 0 d'après l'énoncé : y_test est composé de la classe du pixel, 1 si couleur peau, 0 sin
    TP = 0 # True positif couleur Peau détectée peau
    TN = 0 # True négatif : Couleur non peau détectée non peau
    FP = 0 # False positif : Couleur non peau détectée peau
    FN = 0 # False Négatif : Couleur peau détectée non peau
    # Compute
    for i in range(len(y_test)):
        if (y_predicted[i] == 1) and (y_test[i] == 1): TP += 1
        elif (y_predicted[i] == 0) and (y_test[i] == 0): TN += 1
        elif (y_predicted[i] == 1) and (y_test[i] == 0): FP += 1
        elif (y_predicted[i] == 0) and (y_test[i] == 1): FN += 1
    else :
        print ("Error de binarization")
    return TP, FN, FP, TN
```

```
import pandas as pd
import seaborn as sns
```

```
def draw_cf_Matrix(TP, FN, FP, TN):
    flatten_cf_matrix = [TP, FN, FP, TN]
    cf_matrix = pd.DataFrame([[TP, FN], [FP, TN]], range(2), range(2))
    print(np.array([[TP, FN], [FP, TN]]).reshape(2,2))
```

```
group_names = ['True Pos', 'False Neg', 'False Pos', 'True Neg']
group_counts = ["{0:.0f}".format(value) for value in np.array(cf_matrix).flatten()]
group_percentages = "{0:.2%}".format(value) for value in flatten_cf_matrix / np.sum(flatten_cf_matrix)

labels = ["{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names, group_counts, group_percentages)]
labels = np.asarray(labels).reshape(2,2)
```

```
ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
ax.set_title('Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Classes')
ax.set_ylabel('\nActual Classes');
```

```
## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['True', 'False'])
ax.yaxis.set_ticklabels(['True', 'False'])
```

```
## Display the visualization of the Confusion Matrix.
plt.show()
```

```
print("Check if the array lengths are compatible : ")
print("binaryClassification.shape = ", len(binaryClassification))
print("y_test.shape = ", y_test.shape)
print("-----")
```

```
# Compute the confusion matrix
[TP, FN, FP, TN] = cf_matrix(y_test, binaryClassification)
```

https://colab.research.google.com/drive/14QMxGy_4k8WlaGwJCZVgSQRCPKdVvIpqscroITo?E5WMHWIBXzoQ

5/11

10/04/2022 20:27

TP3_ML_Benslimane.iyyrb - Collaboratory

$$p(x|chair) = \frac{1}{\sqrt{2\pi}\sigma_{Cb}} \exp\left(-\frac{(Cb - m_{Cb})^2}{2\sigma_{Cb}^2}\right) * \frac{1}{\sqrt{2\pi}\sigma_{Cr}} \exp\left(-\frac{(Cr - m_{Cr})^2}{2\sigma_{Cr}^2}\right)$$

Où m_{Cb} , m_{Cr} , σ_{Cb} , σ_{Cr} représentent les moyennes et écarts-type de chacune des composantes. Déterminer m_{Cb} , m_{Cr} , σ_{Cb} , σ_{Cr} .

Utiliser la fonction $\text{norm1}(x, m, \sigma)$ pour estimer la probabilité de tous les pixels de la base d'apprentissage X_train d'appartenir à la teinte chair : $p([Cb Cr] / chair) = \text{norm1}(Cb, m_{Cb}, \sigma_{Cb}) * \text{norm1}(Cr, m_{Cr}, \sigma_{Cr})$ On stockera ces valeurs dans un vecteur p1_train.

Questions

Quelle est la dimension de m_{Cb} , m_{Cr} , σ_{Cb} , σ_{Cr} ?

Pour un pixel x de teinte chair donnée, quelle est la dimension de $p(x|chair)$?

Quelle est la dimension du vecteur p1_train ?

Quelle hypothèse nous permet d'estimer la valeur de la loi normale à partir de l'équation précédente ?

```
mCb = np.mean(T_train[:,0]) # moyennes et écarts-type de la composante Cb
mCr = np.mean(T_train[:,1]) # moyennes et écarts-type de la composante Cr
```

```
σCb = np.std(T_train[:,0]) # écarts-type de la composante Cb
σCr = np.std(T_train[:,1]) # écarts-type de la composante Cr
```

```
print("Les moyennes : \nmCb = ", mCb, "\nmCr = ", mCr, "\nles écarts-types : \nσCb = ", σCb, '\nσCr = ', σCr)
print("La dimension de mCb, mCr, σCb, σCr est de 1")
```

```
def norml(x, m, σ):
    p = 1/(math.sqrt(2*math.pi)*σ)*math.exp(-(x-m)/(2*σ*σ))
    return p
```

```
Les moyennes :
mCb = 104.29577464788733
mCr = 155.19485328813772
Les écarts-types :
σCb = 9.867914388009926
σCr = 11.607693835769547
La dimension de mCb, mCr, σCb, σCr est de 1
```

```
#estimer la probabilité de tous les pixels de la base d'apprentissage X_train d'appartenir
p1_train = [] # On stockera ces valeurs dans un vecteur p1_train
for i in range(T_train.shape[0]):
    p1_train.append(norml(T_train[i, 0], mCb, σCb) * norml(T_train[i, 1], mCr, σCr)) #p ([
#print("\np1_train = ", p1_train)
print("len(p1_train) = ", len(p1_train))
print("len(p1_train) = ", len(p1_train))
print("Pour un pixel x de teinte chair donnée, la dimension de  $p(x|chair)$  est de : 1")
print("La dimension du vecteur p1_train = ", len(p1_train))
```

https://colab.research.google.com/drive/14QMxGy_4k8WlaGwJCZVgSQRCPKdVvIpqscroITo?E5WMHWIBXzoQ

3/11

10/04/2022 20:27

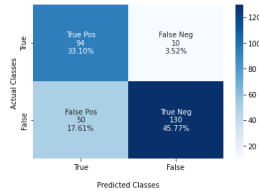
TP3_ML_Benslimane.iyyrb - Collaboratory

```
draw_cf_Matrix(TP, FN, FP, TN)
```

```
Check if the array lengths are compatible :
binaryClassification.shape = 284
y_test.shape = (284,)
-----
```

```
[[ 94 10]
 [ 50 130]]
```

Confusion Matrix with labels



▼ Precision and recall

When we have a **class imbalance**, accuracy can become an unreliable metric for measuring our performance. For instance, if we had 99/1 split between two classes, A and B, where the rare event B, is our positive class, we could build a model that was 99% accurate by just saying everything belonged to class A. This problem stems from the fact that true negatives will be very large, and being in the numerator, they will make the results look better than they are. Clearly we should not bother building a model if it doesn't do anything to identify B. Thus we need different metrics that will discourage this behavior. For this, we use precision and recall instead of accuracy.

Precision is the ratio of true positives to everything flagged positive

$$precision = TP / (TP + FP)$$

Recall : What proportion of actual positives was identified correctly

$$recall = TP / (TP + FN)$$

$$Taut\ de\ bonne\ classification = TP + TN / (TP + TN + FN + FP)$$

https://colab.research.google.com/drive/14QMxGy_4k8WlaGwJCZVgSQRCPKdVvIpqscroITo?E5WMHWIBXzoQ

6/11

```
def ComputeClassifMetrics(TP,FN,FP,TN):
    accuracy = (TP+TN)/(TP+TN+FN+FP)
    precision = TP/(TP+FP)
    recall = TP/(TP+FN)
    sensibillite = TP/(TP+FN)
    specificite = TN/(FP+TN)
    return accuracy, precision, recall, sensibillite, specificite

# Comment estimer le rappel et la précision ?
print("precision = ",TP/(TP+FP) )
print("recall = ", TP/(TP+FN))
# Comment estimer le taux de bonne classification ?
print("Taut de bonne classification = ", (TP+TN)/(TP+TN+FN+FP))

# Pourquoi avoir choisi ce seuil initial ?

precision = 0.6527777777777778
recall = 0.98938461538461539
Taut de bonne classification = 0.7887323943661971
```

Figure 2. Increasing classification threshold.

The number of false positives decreases, but false negatives increase. As a result, precision increases, while recall decreases:

Figure 3. Decreasing classification threshold.

False positives increase, and false negatives decrease. As a result, this time, precision decreases and recall increases:

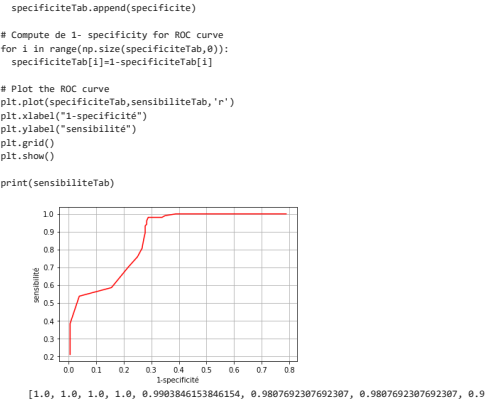
c. Courbe ROC

Plutôt que de choisir un seuil arbitraire, on choisit 20 valeurs de seuil régulièrement réparties entre min(P1_train) et max(P1_train). Pour chaque valeur de seuil, estimer la précision et le rappel et tracer la courbe ROC

```
NB = 20
step = (np.max(p1_train) - np.min(p1_train) ) / NB
SEUILS = np.arange (np.min(p1_train), np.max(p1_train), step)

sensibilliteTab = []
specificiteTab = []

# Estimate + binarize + compute the confusion metrix and classification metrics for each t
for seuil in SEUILS:
    y_predicted = classifyData(p1_test,seuil)
    [TP,FN,FP,TN] = cf_matrix(y_test, y_predicted)
    #print(TP,FN,FP,TN)
    accuracy, precision, recall, sensibillite, specificite = ComputeClassifMetrics(TP,FN,FP,T
    # Append to array
    sensibilliteTab.append(sensibillite)
```



[1.0, 1.0, 1.0, 1.0, 0.9983846153846154, 0.9807692307692307, 0.9807692307692307, 0.9

Déterminer sur la courbe ROC le point de fonctionnement tel qu'il y ait autant de faux positifs que de faux négatifs.

Que vaut le taux de reconnaissance pour ce point ?

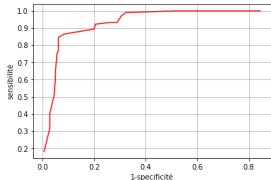
III. Modélisation de la densité de probabilité a priori de la teinte
chaire par une loi normale 2D

Reprendre la partie II en utilisent une loi normale 2D définie par :

$$p(x) = \frac{1}{2\pi \det(\Sigma)} \exp(-0.5(x - m)^T \Sigma^{-1} (x - m))$$

```
plt.show()

print(sensibilliteTab)
```



[1.0, 1.0, 0.9983846153846154, 0.9711538461538461, 0.9326923076923077, 0.93269230769

```
# Transpose the T_train

Sigma = np.cov(np.transpose(T_train))
print("La matrice de covariance : \n",Sigma)
print("\n-----")

# Calcul de la probabilité en chaque point xi de p1_train
p2_train = []
for i in range(T_train.shape[0]):
    p2_train.append(norm2(T_train[i],m,Sigma))

#print("p2_train = ",p2_train)

Le vecteur moyen des donnés : [104.29577465 155.19485321]
La matrice de covariance :
[[ 82.35595391 -89.32551106]
 [-89.32551106 134.94974514]]

# On estime la probabilité sur les points de test
p2_test=[]
for i in range(np.size(X_test,0)):
    p2_test.append(norm2(X_test[i],m,Sigma))

#print(p2_test)

NB = 20
step = (np.max(p2_train) - np.min(p2_train) ) / NB
SEUILS = np.arange (np.min(p2_train), np.max(p2_train), step)

sensibilliteTab = []
specificiteTab = []

# Estimate + binarize + compute the confusion metrix and classification metrics for each t
for seuil in SEUILS:
    y_predicted = classifyData(p2_test,seuil)
    [TP,FN,FP,TN] = cf_matrix(y_test, y_predicted)
    #print(TP,FN,FP,TN)
    accuracy, precision, recall, sensibillite, specificite = ComputeClassifMetrics(TP,FN,FP,T
    # Append to array
    sensibilliteTab.append(sensibillite)
    specificiteTab.append(specificite)

# Compute de 1- specificity for ROC curve
for i in range(np.size(specificiteTab,0)):
    specificiteTab[i]=1-specificiteTab[i]

# Plot the ROC curve
plt.plot(specificiteTab,sensibilliteTab,'r')
plt.xlabel("1-specificité")
plt.ylabel("sensibillité")
plt.grid()
plt.show()
```

Où m et Σ sont la moyenne et matrice de covariance estimées sur les pixels de peau de la base d'apprentissage. On appellera $norm2(x, m, \Sigma)$ la fonction qui renvoie la probabilité en un point x .

```
def norm2(x, m, cov):
    a = np.dot(np.transpose((x-m)), np.linalg.inv(cov))
    a = np.dot(a, (x-m))
    p = 1/(math.sqrt(2*math.pi* np.linalg.det(cov)))*math.exp(-0.5*a)
    return p
```

Tracer la nouvelle courbe ROC et comparer les deux modélisations de la densité de probabilité.

Double-cliquez (ou appuyez sur Entrée) pour modifier

```
# Fonction qui renvoie la probabilité en un point x
def norm2(x, m, cov):
    a = np.dot(np.transpose((x-m)), np.linalg.inv(cov))
    a = np.dot(a, (x-m))
    p = 1/(math.sqrt(2*math.pi* np.linalg.det(cov)))*math.exp(-0.5*a)
    return p
```

Matrice de Covariance

Rappel sur les matrices de covariance

Si on dispose de N vecteurs de données x_i de dimension n , leur matrice de covariance est estimée par :

$$\Sigma = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)(x_i - \mu)^T$$

où μ est le vecteur moyen des données x_i

La matrice de covariance est de dimension $n \times n$. Elle est **symétrique**. Ses valeurs représentent la **forme du nuage de points**

Estimer m et Σ la moyenne et la matrice de covariance sur les pixels de peau de la base
Pixels de peau : T_train

```
mCb = np.mean(T_train[:,0]) # moyennes de la composante Cb
mCr = np.mean(T_train[:,1]) # moyennes de la composante Cr
m = np.array([mCb, mCr]) # Le vecteur moyen des données
#m = m*np.mean(T_train,axis=0)
```

```
print("Le vecteur moyen des donnés :",m)
print("-----")
# La matrice de covariance est de dimension n x n. Elle est symétrique. Ses valeurs repré
```

Syntax: numpy.cov(m, y=None, rowvar=True, bias=False, ddof=None, fweights=None, aweights
Parameters: m : [array_like] A 1D or 2D variables. variables are columns